UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Report 76-0004	2. GOVT ACCESSION NO. (9) Research and	3. RECIPIENT'S CATALOG NUMBER development rept.
4. TITLE (and Subtitle) (6) COMRADE ABSOLUTE SUBROUTINE UTILITY USERS MANUAL,		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Michael A. Wallace		8. CONTRACT OR GRANT NUMBER(s) (16)
9. PERFORMING ORGANIZATION NAME AND ADDRESS David W. Taylor Naval Ship Research and Development Center Bethesda, MD 20084		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63509N Project S0331H Task 14525 Work Unit 1-1850-030
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE (11) January 1976
		13. NUMBER OF PAGES 52
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 49p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) (14) DTNSRDC-76-0004		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) COMRADE; Computer Program; Utility program; library searching; library loading; library sharing		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The COMRADE Absolute Subroutine Utility (ASU) comprises a program that can be used to build and maintain a library of FORTRAN subroutines in executable code, and two FORTRAN-callable subroutines that can be executed to perform library initialization, and loading and execution of library subroutines. Each subroutine on an ASU library is organized as a variable-length page which can be swapped into and executed within a predetermined area of memory. This arrangement allows an application programmer to minimize memory requirements; reduce disk storage; pass parameters to library subroutines; randomly access

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

387682

each subroutine; generate executable code which can be shared by a set of application programs; and modify library subroutines without having to rebuild application programs.

The ASU is currently operational on DTNSRDC's CDC 6700 SCOPE 3.4 computing system.

DEPARTMENT OF THE NAVY

DAVID W. TAYLOR
NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER
BETHESDA, MD. 20084

COMRADE
ABSOLUTE SUBROUTINE UTILITY
USERS MANUAL

by

Michael Wallace

Prepared for
The Naval Ship Engineering Center
Code 6105B

Developed For
CASDAC
The Computer Aided Ship Design and Construction Project

CASDAC No. 520523/APAC

UM-04

January 1976 Report 76-0004

FOREWORD

The COMRADE Absolute Subroutine Utility is a component of the comprehensive Computer Aided Design Environment (COMRADE) software system. The COMRADE software has been developed at the David W. Taylor Naval Ship Research and Development Center and funded by the Naval Ship Systems Command (now a part of the Naval Sea Systems Command) in support of their Computer Aided Ship Design and Construction (CASDAC) effort to facilitate the design and construction of Naval vessels. Although conceived in support of ship design, the COMRADE software has been developed for use as a general design tool. Three separate components are included: the COMRADE Executive System; the COMRADE Data Management System; and the COMRADE Design Administration System. All of these components are operable on the DTNSRDC Control Data 6700 computing system (SCOPE 3.4 operating system) and are documented in a set of eight DTNSRDC reports:

COMRADE - The Computer Aided Design Environment Project, An Introduction
COMRADE Executive System Users Manual
COMRADE Data Storage Facility Users Manual
COMRADE Absolute Subroutine Utility Users Manual
COMRADE Data Management System Primer
COMRADE Data Management System Host Language Interface Users Manual
COMRADE Data Management System Conversational Interface Users Manual
COMRADE Design Administration Users Manual

Inquiries concerning any of these components should be directed to the Computer Systems Development Group, Computer Sciences Division (Code 183), DTNSRDC.

It is understood and agreed that the U.S. Government shall not be liable for any loss or damage incurred from the use of these computer programs.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Many computer users are responsible for solving large complex problems that involve the execution of a number of separate computer programs developed independently by various persons. To enable these independently developed programs and associated input data to be coordinated and integrated into a single easy-to-use system, the Computer Aided Design Environment (COMRADE) software[1 2] has been provided. The COMRADE software is presently operational on the CDC 6700 SCOPE 3.4 computing facility at the David W. Taylor Naval Ship Research and Development Center (Figure 1). COMRADE comprises three major components: the Executive,[3] the COMRADE Design Administration System,[4] and the COMRADE Data Management System.[5 6] Each of the three draws upon subroutines contained in a large FORTRAN library of more than 250 subroutines in all, 70 of which are user-callable. These routines perform various data management maintenance and retrieval functions and file management operations essential to most application programs. Since the solution of large complex problems generally involves the performance of nearly all of the functions provided by the FORTRAN library subroutines, the amount of main core required to store the necessary coding becomes prohibitive. The maximum partition size available to programs operating under the INTERCOM time-sharing system is 60K.[7] To enable the oversized program to comply with the main memory limitations, the user must segment his program into a series of smaller, independently executing units which

---

[1] References are listed on page 43 in the order of their use.

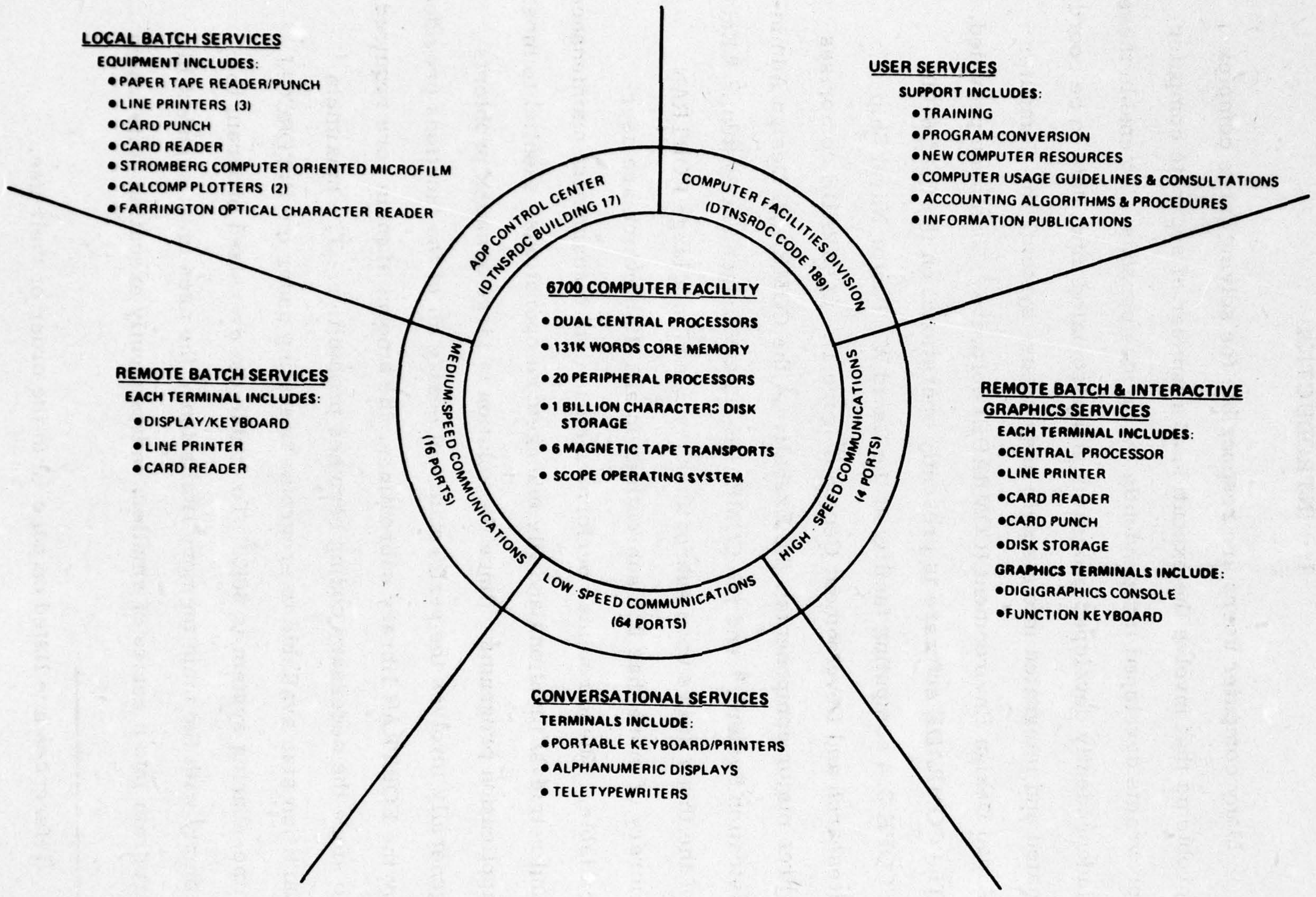
LOCAL BATCH SERVICES
EQUIPMENT INCLUDES:
● PAPER TAPE READER/PUNCH
● LINE PRINTERS (3)
● CARD PUNCH
● CARD READER
● STROMBERG COMPUTER ORIENTED MICROFILM
● CALCOMP PLOTTERS (2)
● FARRINGTON OPTICAL CHARACTER READER
USER SERVICES
SUPPORT INCLUDES:
● TRAINING
● PROGRAM CONVERSION
● NEW COMPUTER RESOURCES
● COMPUTER USAGE GUIDELINES & CONSULTATIONS
● ACCOUNTING ALGORITHMS & PROCEDURES
● INFORMATION PUBLICATIONS
ADP CONTROL CENTER (DTNSRDC BUILDING 17)
COMPUTER FACILITIES DIVISION (DTNSRDC CODE 189)
6700 COMPUTER FACILITY
● DUAL CENTRAL PROCESSORS
● 131K WORDS CORE MEMORY
● 20 PERIPHERAL PROCESSORS
● 1 BILLION CHARACTERS DISK STORAGE
● 6 MAGNETIC TAPE TRANSPORTS
● SCOPE OPERATING SYSTEM
REMOTE BATCH SERVICES
EACH TERMINAL INCLUDES:
● DISPLAY/KEYBOARD
● LINE PRINTER
● CARD READER
MEDIUM-SPEED COMMUNICATIONS (16 PORTS)
HIGH-SPEED COMMUNICATIONS (4 PORTS)
LOW-SPEED COMMUNICATIONS (64 PORTS)
REMOTE BATCH & INTERACTIVE GRAPHICS SERVICES
EACH TERMINAL INCLUDES:
● CENTRAL PROCESSOR
● LINE PRINTER
● CARD READER
● CARD PUNCH
● DISK STORAGE
GRAPHICS TERMINALS INCLUDE:
● DIGIGRAPHICS CONSOLE
● FUNCTION KEYBOARD
CONVERSATIONAL SERVICES
TERMINALS INCLUDE:
● PORTABLE KEYBOARD/PRINTERS
● ALPHANUMERIC DISPLAYS
● TELETYPEWRITERS

Figure 1 - DTNSRDC Control Data 6700 Computing Facility

will fit within the main memory space provided. The Absolute Subroutine Utility (ASU) described in this report supplies the user with the software he needs. The ASU supplies

(a) a program that can be used to build and maintain a library of subroutines in absolute code, and

(b) a set of two user-callable subroutines that can carry out library initialization and load and execute functions.

Each of the ASU subroutines is organized as a variable length page which can be swapped in to and out of a single predetermined area in core memory when called. This arrangement allows the application programmer to accomplish the following tasks:

. Minimize core requirements,

. Reduce amount of disk storage space needed,

. Pass parameters between library subroutines,

. Randomly access each subroutine,

. Generate absolute subroutine code which is useable by all application programs, and

. Modify an ASU subroutine without having to reload any application programs.

The ASU may be used in conjunction with the DTNSRDC SCOPE 3.4 overlay facility to provide an efficient means of maintaining and executing the many user-callable subroutines commonly required for the solution of large complex problems.

II. PHILOSOPHY OF THE ABSOLUTE SUBROUTINE UTILITY

THE ROLE OF THE UTILITY

The role of the ASU software is twofold: First, it provides a technique for constructing and organizing large applications so that they will fit within the space allocated in main memory; and second, it provides a mechanism for generating and maintaining libraries of subroutines in absolute form, thus obviating the need for reconstructing an application program whenever a subroutine is modified.

The first function is accomplished by swapping subroutines in to and out of a single predefined area in main memory. These subroutines are stored in absolute form in a randomly organized library. The swapping mechanism provided is a dynamic, FORTRAN-callable loader capable of both loading and executing the designated subroutines. This loader consists of two user-callable subroutines, INITLDR and COMRLDR. INITLDR initializes a library of subroutines stored in absolute form and then reads the file directory into a common array. The file directory, contains the memory address of the subroutine swap area and the library file addresses. COMRLDR, called after INITLDR has executed, reads and executes the subroutine specified. Before COMRLDR begins the read operation, it checks to see whether the specified subroutine is already resident in memory. If so, COMRLDR performs a simple transfer operation; otherwise it issues a single read operation, which causes the specified subroutine and all of its subprograms to be read into the swap area, and then it performs a simple transfer operation.

The generation of a new library of subroutines in absolute form is accomplished by an ASU program called ASLGEN (for Absolute Subroutine Library Generator). Card input supplies ASLGEN with the following information: data necessary for computing the address in memory of the area to be shared by the executing subroutines; names of each of the subroutines; and the permanent file name for the newly generated library. After ASLGEN has finished executing, the newly produced Absolute Subroutine Library (ASL) is automatically cataloged into the SCOPE permanent file system.

ASLGEN processes each of the subroutines to be included in a given ASL, one by one. The names of the subroutines to be processed are entered as input data. Each specified subroutine is stored as a variable-length logical record in the ASL which may include as many as 250 subroutines. A member subroutine typically will call upon several subprograms, and will require the use of one or more labeled common block areas. Since each ASL member is stored in absolute form, each logical record will contain both a "master" subroutine (given an assigned name) and whatever "slave" subprograms the master subroutine requires. The initial values of each of the labeled common block areas required by the various ASL member subroutines are stored in a separate logical record which is read into memory by INITLDR; the values and lengths of the labeled common block areas are specified when the ASL is generated. All common block areas to be used must be initialized before the ASL subroutines that will use them are executed.

THE SWAPPING MECHANISM

INITLDR stores the memory address of the swap area designated for that ASL file, reads in the direc ory of subroutine names and addresses associated with that file, and initializes any local (known only to ASL subroutines) common block (described later) areas to be used. COMRLDR is called to load and execute ASL subroutines. If the subroutine specified is not yet resident in the swap area, COMRLDR performs a binary search of the names and addresses of the ASL directory to find the entry that corresponds to that subroutine name, and then issues a read instruction to bring the appropriate logical record (containing both the master program and the slave subprograms) into the prespecified memory location. When this is done, COMRLDR issues a COMPASS assembly language "return jump" instruction[8] (a transfer) to the master subroutine which is located immediately behind any local labeled common block areas. The master and slave subprograms then begin executing, reading and writing data into local labeled common areas as needed.

When the master program has completed executing, it returns control to COMRDLR, and COMRLDR relinquishes control to the calling program. The calling program may then call COMRLDR to load and execute another ASL subroutine. This procedure may be repeated as often as necessary until all required ASL subroutines have been executed. Since the local labeled common block areas are preserved in memory, the various master subroutines can communicate with each other through the contents of the local common areas.

LIBRARY FILE GENERATION

The ASU program ASLGEN is used to generate library files of absolute subroutines. It exists as a permanent file and is attached by name, the word ASLGEN causing the appropriate SCOPE control cards to be entered for execution. The last control card in the series reexecutes ASLGEN. User control cards included behind the ASLGEN card are saved and resubmitted when ASLGEN has terminated.

Subroutine names, loadset directives,[9] FORTRAN labeled common block statements,[10] and user EDITLIB libraries[11] [12] are supplied to ASLGEN as input. As many as five EDITLIB libraries may be specified in the first five parameters of the SCOPE control card; at least one must be specified. The name of the input file to be used by ASLGEN is indicated in the sixth parameter. If an input file name is not indicated, the default parameter INPUT is used.

ASLGEN produces two permanent files— an ASL file and a partitioned data set (PDS)[13] file—and a print file, the PDS file being used to facilitate updates to the ASL subroutines. The print file traces each of the jobsteps performed in generating an ASL, each jobstep consisting of a FORTRAN compilation, a system load, and a PDS "ADD" directive. One jobstep is performed for each subroutine that is included in the ASL as a member subroutine.

The final step performed by ASLGEN is the transformation of the newly generated PDS into an ASL, using the program ABSFILE.

The flow of control through ASLGEN is illustrated in Figure 2. Discussions of some of the more significant functions of ASLGEN follow.

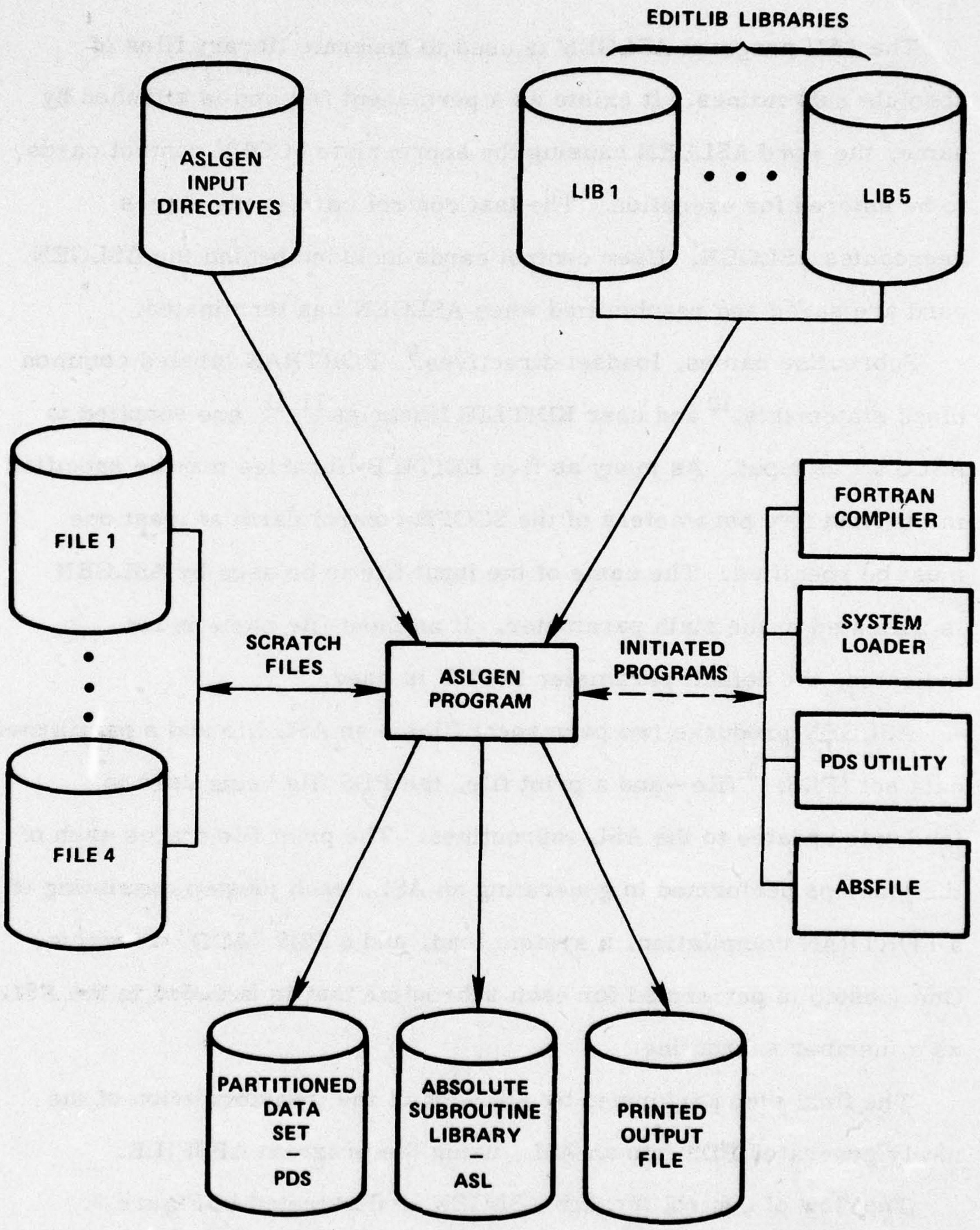

Figure 2 - Flowchart of ASLGEN

Processing Labeled Common Block Areas

Typically, an ASL member subroutine calls on several subprograms and uses several labeled common block areas. The locations of these areas — called local common block areas to distinguish them from the common block areas used by the program calling COMRLDR — are known only to the ASL subroutines and are used for passing parameters among the different subroutines. The length and definition of local labeled common must be the same for all members of a particular ASL. This requirement ensures that all of the ASL subroutines will execute in exactly the same segment of the swap area reserved by the calling program.

Under the Control Data SCOPE 3.4 operating system, loading of a program will begin at the relative memory location 101B (101 octal) of the memory space allocated for that program. Typically, the user application will define its ASL swap area as beginning in location 101B by inserting a FORTRAN labeled common statement immediately after the PROGRAM statement; the ASLGEN program will normally generate a library of absolute subroutines that expect to be loaded and executed beginning in location 101B. This procedure is the simplest and most straightforward use of the ASU software, although the user application may specify a starting location of the ASL swap area at an address other than 101B if he will first notify ASLGEN of that fact so that the ASL it generates will have the correct origin or initial load address.

The FORTRAN labeled common statement provides the mechanism needed for identifying and defining each local common block area for a particular ASL, and for altering the initial load address. By specifying a set of these statements in the input stream for ASLGEN, the user can define all of his local common areas. The first such statement in the set may in fact be a dummy statement used simply to "push down" the initial load address for an ASL; the length of the dummy area must be the difference between location 101B and the memory address of the swap area indicated in a SCOPE system load map. Since the swap area is typically a FORTRAN labeled common block area, its relative memory address is easily recognized in the map.[9]

When the initial load address is to be "pushed down" from 101B, the user must supply an additional input value to the ASLGEN program. This piece of information — identified as the NUM input parameter — is explained in detail in Section III under the heading "The Absolute Subroutine Library Generator."

Processing Within an ASLGEN Jobstep

A jobstep — consisting of a FORTRAN compilation, a system load, and a partition data set ADD directive — must be executed for each subroutine to be included in an ASL. The ASLGEN program causes the SCOPE control cards appropriate for each of these functions to be submitted for execution.

During the execution of each ASLGEN jobstep, a SCOPE 3.4 absolute overlay file is created which contains a master ASL subroutine and any required slave subprograms. Since a single ASL may contain as many as 250 master subroutines, it is possible to have 250 absolute overlay files generated during a single execution of ASLGEN. Since that many files will not be allowed at a DTNSRDC SCOPE 3.4 control point, a PDS — an intermediate file — is established during the construction of an ASL file. As an absolute overlay file is created, it is added to a PDS as a partition and then unloaded, thereby keeping the number of active files to a minimum. Each partition is named for one of the subroutines to be added to the ASL library. The partition names make up the subroutine list in the ASL directory of subroutine names and file addresses. The PDS file is the file used in generating an ASL.

ASLGEN can both create a new PDS and, if necessary, cause the partitions on an existing PDS to be modified. In either case, a FORTRAN program is created and then compiled for each master subroutine (designated by name in the input to ASLGEN) that is to be included in the new ASL. The FORTRAN program consists of an OVERLAY statement, a PROGRAM statement, any labeled common block statements that were specified in the ASLGEN input stream, a CALL statement for the master subroutine, and an END statement.

After the object code for the program PROGRAM has been generated, it is then loaded — using the loadset directives specified in the ASLGEN input stream — and bound — using the SCOPE system linking loader and the EDITLIB libraries named in the ASLGEN input stream — so that all external references (calls to "slave" subprograms) will be resolved. It is at this time that any local common areas needed are initialized, using BLOCK DATA subprograms that may reside on the EDITLIB libraries. Following the resolution process, a file of absolute code is created. It is this file which is then added as a partition to the intermediate PDS file by way of the PDS ADD directive.

An example of a FORTRAN program created by ASLGEN for the master subroutine ADDCOM is shown in Figure 3. The common block statements for blocks MISC, C832, C832I, USCA and BLOCK1 are supplied in the input stream for ASLGEN and are included in each of the absolute overlays created during an ASLGEN jobstep. This practice guarantees that the length and definition of all local labeled common for the ASL, as well as the load addresses for each master subroutine and any of its slave subprograms, will be identical.

```
OVERLAY (PROG, 0, 0)
PROGRAM MAIN
COMMON/MISC/IDUM1(3)
COMMON/BLOCK1/IDUM2(462)
COMMON/C832/IDUM3(63)
COMMON/C832I/IDUM4(600)
COMMON/USCA/DUM5(S)
CALL ADDCOM
END
```

Figure 3 - A Sample FORTRAN Program Created by ASLGEN

Notice in Figure 3 that since the source code defines a main program, no FORTRAN RETURN statement is included. Hence, the COMPASS assembly language subroutine END. must be appended to each FORTRAN program created by ASLGEN. The END. subroutine is assembled (compiled with the main program) and subsequently loaded into the absolute overlay file created during an ASLGEN jobstep. After a master subroutine has finished executing, the END. subroutine is called to ensure a return to the COMRLDR subroutine. A more detailed explanation of the subroutine END. is presented in Appendix A.

Transformation of a PDS File into an ASL

The final step in the preparation of an ASL is the transformation of the intermediate PDS into a new ASL by the program ABSFILE. The control card specifying execution of ABSFILE is created and submitted to the control card buffer by ASLGEN. After ABSFILE has finished executing, the newly generated ASL is automatically cataloged by ASLGEN.

The transformation of the PDS file into an ASL involves three operations. First, to enable the use of a binary search technique during retrieval, the PDS directory is transformed into a sorted ASL directory of master subroutine names and file addresses. Second, all local labeled common block areas stored with each partition of the PDS file are stripped out and stored on the ASL file as a single logical record; any dummy FORTRAN labeled common block statement used to push down the initial load address for the ASL subroutines — and any secondary storage space required by that dummy statement — will be ignored, and relevant local common areas will be copied onto the ASL. Third, each partition is written to the ASL file, transforming the main program into a subroutine.

Before transforming the PDS, ABSFILE will request permanent file space for the resultant ASL file. After the transformation, ABSFILE will catalog the new ASL, using the name specified in the input stream for ASLGEN. An old PDS file that is to be modified and used to generate a new ASL will be extended (an old PDS file being one which existed prior to the current execution of ASLGEN).

Modifications to an old PDS typically will consist of deletions of old master subroutines and additions of new like-named master subroutines.

After transformation, an ASL library file will contain the following information:

- A 255-word header record
- A variable-length local labeled common block record (if local common exists)
- As many as 250 variable-length records of absolute code, each record made up of a master subroutine and whatever slave subprograms are required

The structure of an ASL file is indicated in Figure 4.

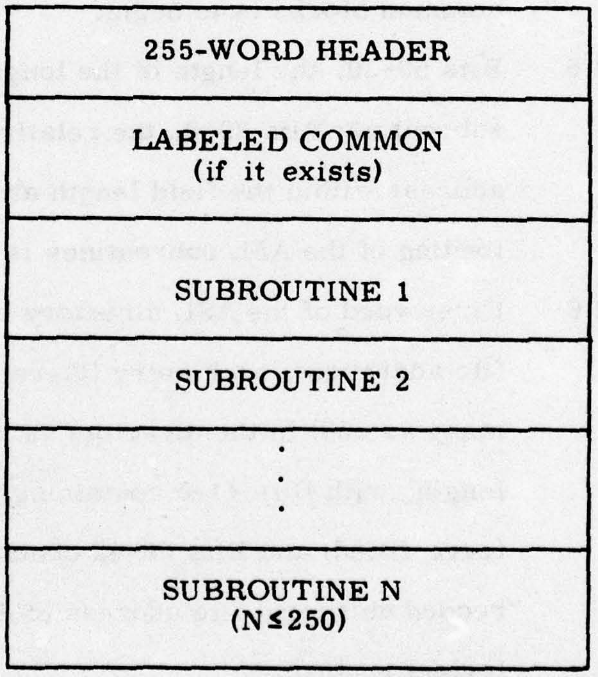

Figure 4 - Structure of an ASL File

The file header will contain the following information:

Word 1	The date the ASL was created
Word 2	The time the ASL was created
Word 3	Bits 59-30, the number of binary searches required to locate a specified name in the ASL directory of names and addresses; Bits 29-0, the number of member subroutines on the ASL.
Word 4	Bits 59-30, the length, in words, of the local labeled common block areas for this ASL; Bits 29-0, the relative memory address within the field length at which loading of the local common blocks is to begin.
Word 5	Bits 59-30, the length of the longest ASL subroutine; Bits 29-0, the relative memory address within the field length at which the loading of the ASL subroutines is to begin.
Word 6	First word of the ASL directory of names and file addresses; each entry (there may be as many as 250) in the directory is one word in length, with Bits 41-0 containing a subroutine name (zero-filled) and Bits 59-42 containing the physical record unit (pru) file address of that subroutine's logical record.

COMPARISON OF THE ASU AND THE CDC SCOPE OVERLAY

The ASU was developed to provide an efficient means of maintaining and executing user-callable subroutine packages that are needed by suites of large application programs. The ASU is used in place of the SCOPE operating system's overlay feature[9 10] for these subroutine packages. The program may use both the ASU and the SCOPE overlay.

The ASU and the SCOPE overlay feature both allow the user to make more efficient use of his allocated memory space, and both require that the user concern himself with bound programs. However, the two capabilities differ in a number of ways, as outlined in the following paragraphs.

Response Time

An absolute subroutine library (ASL) is a randomly organized file of member subroutines whose directory is searched by way of a binary search technique. An overlay file is sequentially organized, and is searched in an end-around manner.

Program Maintenance

Typically, certain subroutines in a given subroutine package will be used by many large application programs in the same suite of applications. If the SCOPE overlay feature is used to handle the programs, one or more copies of each common subroutine must be included in each file; thus if one of these subroutines is changed, all of the files using that subroutine have to be re-created. Under the ASL library concept, however, a single copy of each subroutine is shared by all applications, so any update is automatically made available to all applications; no program need ever be reconstructed unless the predefined swap area becomes too small for the longest ASL subroutine used.

File Storage

As mentioned previously, use of the overlay feature with applications having common subroutines requires that at least one copy of each common routine be included on every file which stores the calling applications. The ASU concept enables a single set of subroutines to be shared by all calling applications. Even if the ASL library file itself becomes very large (due to the duplication of slave subprograms in the file's various logical records), it is usually still smaller than the area that would otherwise be required to store all the overlayed common subroutines.

Data Communication

In the overlay scheme, data elements are passed between application program overlays through labeled or blank common areas. Any element of a labeled or blank common block declared in the main overlay may be referenced by any primary or secondary overlay. Any labeled or blank common declared in a primary overlay may be referenced only by that primary overlay and its associated secondary overlays, never by the main overlay.

In the ASU scheme, data elements are passed to ASL library subroutines as conventional FORTRAN parameters. Hence, there is no need to buffer data as would be the case if an ASU subroutine in a higher level overlay had to communicate with data in a lower level overlay.

Standardization

An ASL library of subroutines provides a standard methodology for constructing and organizing suites of large application programs. Such an approach offers a better environment for learning, an important factor for programmers who are shifted from one application to another to meet scheduled commitments. Moreover, the ASU approach reduces communication problems between the group responsible for maintaining the ASL and the group, perhaps in a different organizational unit, responsible for implementing the suites of ASU applications.

Load Time

The execution time required to load and link large overlayed programs into bound units is usually quite extensive. In some instances, the same subprograms will be included within several overlays of a single program file, making multiple linking operations necessary.

Under the ASU library concept, only the nine subprograms of the loader portion of the ASU software (Appendix B) have to be linked with the calling application.

SUMMARY

In comparison to the SCOPE overlay feature, the ASU provides a more efficient means of handling suites of large application programs that call upon subroutine packages requiring significant memory space. The ASU facilities enable absolute subroutine libraries to be created, maintained, and processed in a way that alleviates some of the problems normally associated with the development of large FORTRAN applications under the SCOPE operating system. The utility is especially helpful during construction of large complex interactive application systems. The ASU does not obviate the need for overlay. Rather, it provides a powerful tool for use in those situations in which a large subroutine package is to be used by many separate application programs.

III. USE OF THE UTILITY

THE SWAPPING MECHANISM

In the following descriptions of the dynamic, FORTRAN-callable, ASU loader subroutines INITLDR and COMRLDR, several coding conventions have been observed. For example, underlined parameters indicate user-supplied values, and overlined parameters indicate values to be returned by the loader subroutines. Standard FORTRAN conventions regarding variable type are observed. For a complete description of the FORTRAN extended programming language normally used with the SCOPE operating system, consult the Control Data documentation.[10] Appendix B gives information concerning memory space requirements for INITLDR and COMRLDR.

Before calling the ASU loader subroutines, the user must reserve a specific area in memory in which the various ASL subroutines can execute. The area reserved must subsume the area in which the subroutines expect to execute. If not, an error will not be detected and any valid user code existing in the space will simply be overwritten by the ASL subroutine code; the COMRLDR subroutine will load into the load address that was specified initially for the generation of the ASL. The initial load address for most ASL's is at location 101B. Typically, a FORTRAN labeled common statement is used to reserve a swap area. Thus the labeled common statement

```
COMMON/AREA/ISPACE(10000B)
```

might be used to reserve a swap area of 10000 octal locations. If more than one ASL library is to be used by a program, and if the subroutines for each library require different areas, a swap area for each library must be reserved. It is possible for the different libraries to share the same swap area.

A schematic of the structure of main memory for a typical ASL application is provided in Figure 5.

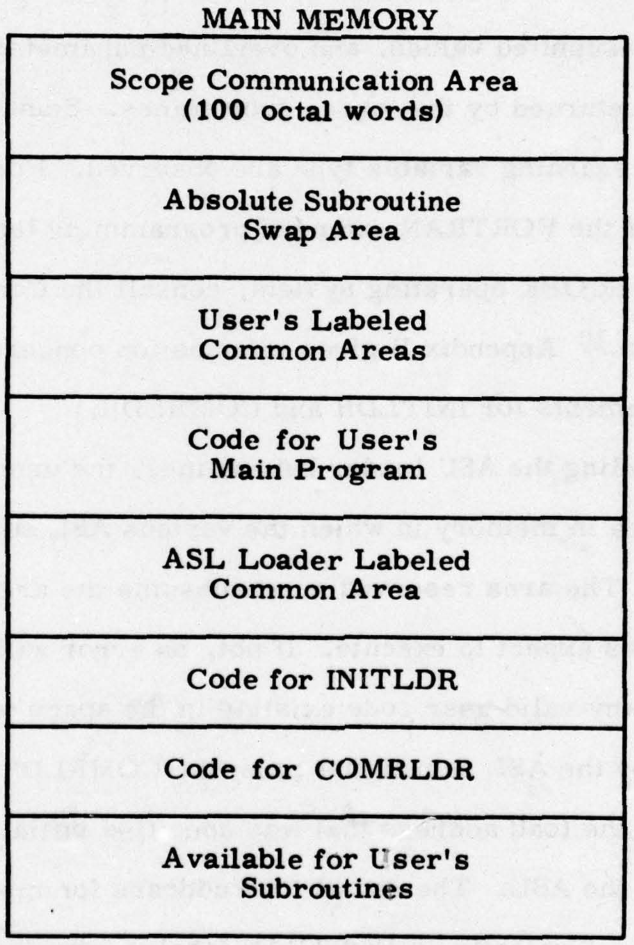
MAIN MEMORY
Scope Communication Area (100 octal words)
Absolute Subroutine Swap Area
User's Labeled Common Areas
Code for User's Main Program
ASL Loader Labeled Common Area
Code for INITLDR
Code for COMRLDR
Available for User's Subroutines

Figure 5 - Structure of Memory for a Typical ASL Application

There is one restriction on the use of FORTRAN input/output for subroutines that are to be executed by COMRLDR. All FORTRAN I/O instructions — such as READ, WRITE, REWIND, ENDFILE -- must be coded using variable file or logical unit numbers. Thus, for example, a READ operation to be performed on a file identified as TAPE5 could be accomplished by the instructions

```
LUN=5
READ(LUN, 100) ...
```

Similarly, the instructions

```
I=20
REWIND I
```

could be used to request the rewinding of TAPE20.

This restriction exists because of the manner in which FORTRAN handles file I/O.

The explicit use of logical unit numbers in I/O statements produces external references that could not be linked to the files defined in a user program. Thus any I/O operation will abort unless variable file or logical unit numbers.

1. INITLDR SUBROUTINE

INITLDR(<u>LFN</u>, <u>NAME</u>, $\overline{\text{IERR}}$)

The INITLDR subroutine must be executed before COMRLDR can be called. INITLDR reads in the 255-word ASL file header and the variable-length logical record containing any local labeled common blocks. Values in the local common blocks are then available for use by any ASL subroutine. Such values are preset, using BLOCK DATA subprograms, whenever an ASL is generated.

INITLDR may be called from an executing program as many times as needed. Local labeled common — if it exists — is initialized each time INITLDR is called. The ASL loader subroutines may be used with the overlay capability of the operating system. When the calls to COMRLDR are scattered amongst many different higher-level overlays, both the swap area and INITLDR must be stored in the main overlay. The swap area may be overlayed; the only general restriction is that INITLDR must be called to initialize the ASL file each time an overlay containing the swap area is entered.

The user may specify an ASL subroutine to be executed if the program should terminate abnormally. The SCOPE system subroutine RECOVR[10] allows a terminated program to obtain a small amount of additional execution time — for use in closing a file or writing a crucial message — before requiring it to give up its control point. The user must merely identify the ASL subroutine to be loaded and executed (by indicating it in the NAME parameter for INITLDR). No provision exists, however, for passing parameters to the subroutine.

Parameters:

LFN	Logical file name of the ASL (left-justified, zero- or blank-filled)
NAME	The name of an ASL subroutine (left-justified, zero- or blank-filled) that is to be called should the calling program terminate abnormally; if no subroutine is to be called, NAME must be set to zero on input.

IERR	An error code; a value of 0 implies a successful INITLDR operation. If IERR is returned with a value of one, LFN is improperly organized and is not a legitimate ASL

2. COMRLDR SUBROUTINE

COMRLDR(SUBNAM, PARM1, PARM2, ..., PARMn)

The COMRLDR subroutine is responsible for loading and executing user-specified ASL subroutines. If a specified routine is already in memory, COMRLDR will simply cause that subroutine to be executed. If the name of the subroutine executed by COMRLDR does not exist in the ASL directory of member names and file addresses, a message, INVALID SUBROUTINE NAME, is written to the program's dayfile and the job aborted. If the ASL file has not been initialized, the message, INITLDR NOT CALLED, is written to the dayfile and the job aborted.

Parameters:

SUBNAM	The name of the master ASL subroutine which is to be loaded (along with all of its slave subprograms) and then executed (left-justified, zero- or blank-filled)
PARMS	The appropriate input and output parameters for SUBNAM; the number of PARM's will vary depending on the requirements of the various SUBNAM's

THE ABSOLUTE SUBROUTINE LIBRARY GENERATOR

The ASLGEN program reads in subroutine names, loadset directives, and labeled common statements, and then generates an ASL, using user-specified EDITLIB libraries, that is automatically cataloged under the permanent file name specified by the user in the ASLGEN input stream.

The structure of a typical ASLGEN job deck containing a single EDITLIB library (identified as LIB) is shown in Figure 6. Operational requirements of ASLGEN are discussed in Appendix B. Figure 7 indicates the structure of a deck of ASLGEN input directives to be included in an ASLGEN Job Deck.

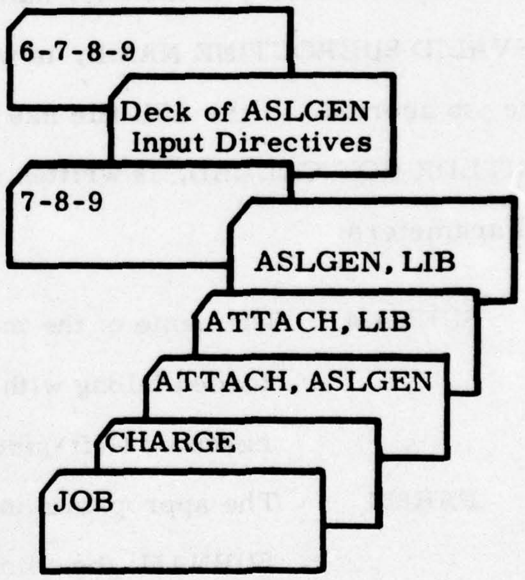

Figure 6 - Structure of a Typical ASLGEN Job Deck

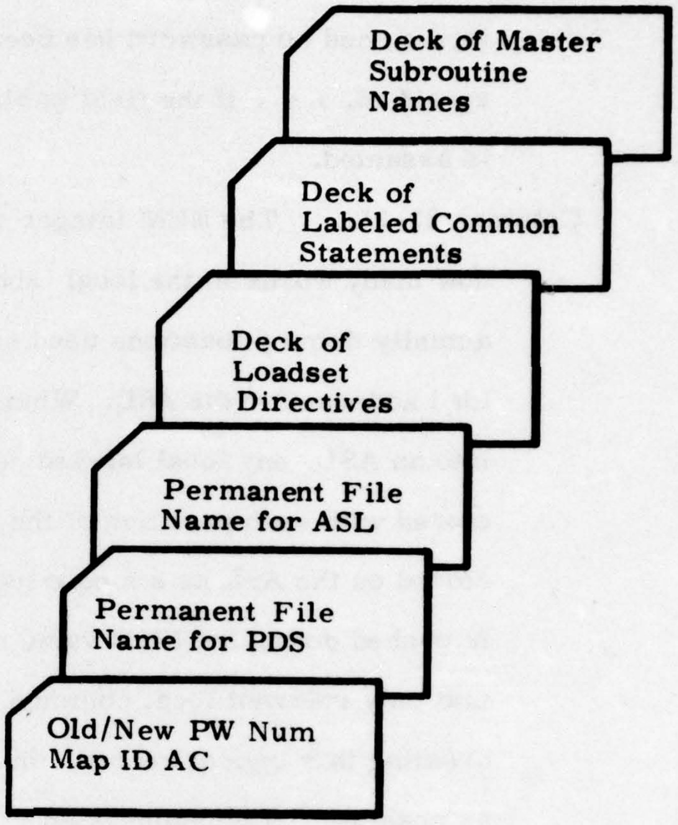
Deck of Master Subroutine Names
Deck of Labeled Common Statements
Deck of Loadset Directives
Permanent File Name for ASL
Permanent File Name for PDS
Old/New PW Num Map ID AC

Figure 7 - Structure of the Deck of ASLGEN Input Directives

Detailed explanations of each of the ASLGEN input directives follow.

Card One (Required)

Columns 1-3: The word OLD or NEW. OLD indicates that an existing partitioned data set is to be updated and extended during the current ASLGEN run; NEW indicates that a new partitioned data set is generated and cataloged.

Columns 11-19: The password or PW value. This value indicates the extend, control, and modify permissions accorded the partitioned data set and ASL files. If PW is equal to NONE, it is assumed no password has been supplied. If no value is specified, i.e., if the field is blank, the password COMRADE is assumed.

Columns 21-25: The NUM integer value. This value indicates how many words in the local labeled common block areas are actually dummy locations used simply to push down the initial load address for the ASL. When the PDS is transformed into an ASL, any local labeled common block areas that are stored with each partition of the PDS are stripped out and stored on the ASL as a single logical record. When the origin is pushed down, the NUM value must be provided to ensure that only relevant local common block locations are used in creating this logical record, thus keeping the record as small as possible. If no value is specified, zero is assumed.

Columns 31-34: The MAP value. This value indicates the type of system load map generated during the compile and load job-steps of ASLGEN. If MAP=OFF, no load map output will be produced. If MAP=ON, then a full load map including all cross references to all entry points will be produced. The default value for MAP is PART. If the default is allowed, a map consisting of the starting addresses of the subprograms will be generated.

Columns 41-44: The ID value. This value indicates the four-character user identifier to be used when cataloging the PDS and ASL files. If the field is left blank, the value will be taken from the charge card of the ASLGEN job deck.

Columns 51-60: The AC value. This value indicates the ten-character account number to be used when the PDS and ASL files are cataloged. If no value is specified, the value will be taken from the charge card.

Card Two (Required)

Columns 1-40: Permanent file name of the partitioned data set used to generate the new ASL.

Card Three (Required)

Columns 1-40: Permanent file name of the new ASL.

Deck of Loadset Directives (Optional)

Each loadset directive needed in the building of each of the ASL subroutines must be included. As many as ten directives may be specified.

Deck of FORTRAN Labeled Common Statements (Optional)

As many as ten FORTRAN labeled common statements necessary for the defining of all local labeled common may be included. The first of these statements may be used to push down the initial load address for the ASL.

Deck of Master Subroutine Names

As many as 250 cards may be included in the deck, one subroutine name per card. A subroutine name must be specified in card columns 1 to 7. At least one card is required.

Before executing the program identified as ABSFILE — a program which transforms an intermediate PDS file into an ASL — ASLGEN must successfully complete all of its previous jobsteps. As described in Section II under the heading "Library File Generation," each of these jobsteps consists of a FORTRAN compilation, a system load, and a PDS "ADD" directive. Prior to executing any of these jobsteps, however, ASLGEN must analyze all of the directives provided in the input stream; several fatal error conditions may be uncovered during the analysis. If an error condition exists, a concise message describing the condition will be written out on the user's output file. The following error conditions may be indicated:

. NO EDITLIB LIBRARY

. OLD OR NEW NOT SPECIFIED FOR PDS FILE

. MORE THAN 250 SUBROUTINE NAMES SPECIFIED

. TOO MANY LOADSET DIRECTIVES

. TOO MANY FORTRAN LABELED COMMON STATEMENTS

. OLD PDS FILE DOES NOT EXIST OR IS UNAVAILABLE FOR UPDATE

After successfully completing all previous jobsteps, ASLGEN executes the ABSFILE program. ABSFILE transforms a PDS file — created by ASLGEN — into an ASL. ABSFILE will request permanent file space for the resultant ASL. It will then build a zero header record, copy any local labeled common block areas (remembering that certain dummy locations as specified by the NUM value are to be ignored), copy the master subroutines (and their slaves), and then rewrite the header with all the appropriate information. Fatal error conditions may be encountered during the execution of ABSFILE. The following error messages are produced. (As with ASLGEN, error messages are written to the output file)

. INCONSISTENT LENGTH OF LOCAL COMMON

. BAD VALUE FOR NUM PARAMETER

. TOO MANY PARTITIONS ON PDS FILE

The first error condition may arise when an old partitioned data set is being modified prior to the generation of a new ASL. If the lengths of the local common block areas in the absolute subroutines being generated are different from the lengths of the local common areas already stored on the PDS, then an inconsistency exists and no new ASL will be generated.

The second condition will arise if a bad value for the NUM parameter is specified in the deck of ASLGEN directives. If the user is pushing down the initial load address for his ASL, he must supply a NUM value indicating exactly how many dummy words are used simply to obtain the desired load address. If the number specified is larger than the actual number of words of labeled common, an inconsistency exists and no ASL will be generated.

The third error condition will occur if the user attempts to add more than 250 subroutines to an ASL.

Once generated, a new ASL becomes available for immediate use, either by a single executing program or by several executing applications executing concurrrently, as depicted in Figure 8. In some instances, the swap area for such a suite of applications may be quite large, perhaps containing of as many as 30000 octal locations, so that the costs associated with the storage of these files will be very high, particularly if the files must be maintained on-line at all times. Appendix C describes a special program, identified as XFORM, which may be used to remove and restore the swap area in a given set of ASL program files so that significant reductions in space and costs for on-line ASL files can be achieved.

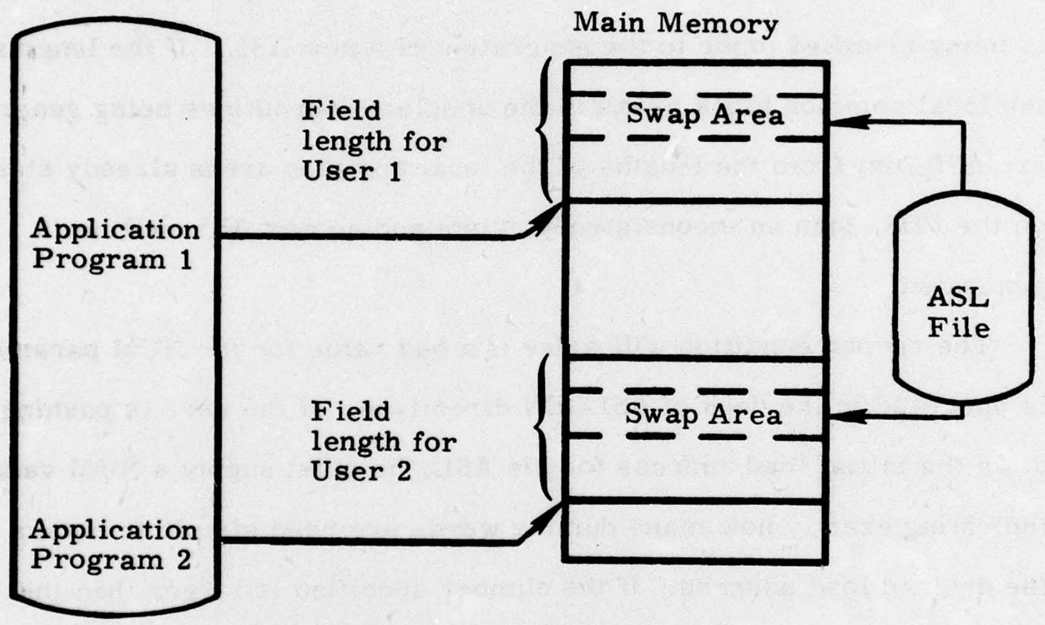

Figure 8 - Concurrent Use of an ASL File

ACKNOWLEDGMENTS

The author wishes to acknowledge Mr. William C. Gorham and Thomas R. Rhodes for their technical contributions during the development of the Absolute Subroutine Utility concept.

APPENDIX A

SUBROUTINE END., A COMPASS SUBROUTINE

As mentioned in Section II under the Heading "Processing Within an ASLGEN Jobstep", a special COMPASS assembly language subroutine, identified as END., is appended to each FORTRAN program created by ASLGEN. This special subroutine is assembled and loaded into each absolute overlay file that eventually becomes a member of an ASL library. By including the END. routine, the user is ensuring the return of control from a master subroutine to subroutine COMRLDR. A second benefit from the use of END. as a replacement for the SCOPE system END. routine is that it prohibits loading of the SCOPE Q8NTRY. subroutine. Since Q8NTRY is not needed during creation of the ASL, library space otherwise requird to contain it and the various subroutines it calls upon can be saved.

The source code for END. follows:

```
         IDENT    END.
         ENTRY    Q8NTRY
         ENTRY    END.
Q8NTRY.  BSS      0
END.     BSS      0
         JP       =XMAIN
       . END
```

APPENDIX B

OPERATING REQUIREMENTS FOR THE ABSOLUTE SUBROUTINE UTILITY

THE SWAPPING MECHANISM

The swapping mechanism consists of nine subroutines, seven of which are called by the two user-callable subroutines INITLDR and COMRLDR. Approximately 1100 octal words of main memory are required to store these nine subroutines which are available in EDITLIB form on the DTNSRDC 6700 SCOPE permanent file, SUBLIB, ID=COMR. The following list indicates the octal size of each loader subroutine, the subroutines called by each, and the number of times each is called. The notation

A(D, 2040) B(2), C(1)

would indicate that Subroutine A, which has an entry point D, is 2040 octal words long, and calls Subroutine B twice and Subroutine C once.

FORTRAN SUBROUTINES

COMRLDR(121)	ABORT(1), FILL(1), JUMP(1), RWPAGEA(1)
FILL(22)	(None)
INITLDR(107)	FILL(2), LWAR(1), REPR(1), RWPAGE(1), RWPAGEA(1)
REPR(LWAR, 26)	COMRLDR(1)
RWPAGE(RWPAGEA, 162)	ZIO(1), FILL(1)

COMPASS SUBROUTINES

ABORT(4)	REQPP(1)
JUMP(17)	(None)
REQPP(10)	(None)
ZIO(33)	REQPP(1)

LABELED COMMON

COMRADE(406)

THE ABSOLUTE SUBROUTINE LIBRARY GENERATOR

The absolute subroutine library generator program ASLGEN requires 50K octal locations in which to execute. Typically, each member subroutine to be added to an ASL library will require between three and five seconds of central processor time.

APPENDIX C

THE PROGRAM XFORM

The program XFORM is used to reduce the file space required to store absolute overlay application programs on the DTNSRDC SCOPE 3.4 system. It does this by removing any labeled common blocks from those files and then adding a small bootstrap program that will subsequently restore these common blocks—and hence the program's internal structure—when the file is to be loaded and executed. Currently, XFORM can remove labeled common areas only if the initial load address for those areas is at memory address 101B.

When absolute overlay files contain large labeled common areas, the use of XFORM can result in significant reductions in on-line file storage. XFORM is especially useful, for example, in reducing the file size of application programs which make use of the ASU and which possess large swap areas for execution of the absolute subroutines.

XFORM is stored in the DTNSRDC SCOPE 3.4 file system as a permanent file XFORM, ID=COMR. The structure of a typical XFORM job deck, including an absolute overlay program file identified as PROGFILE which has a labeled common area of 1000B locations in length, is shown in Figure 9. XFORM requires 45K octal locations in which to execute.

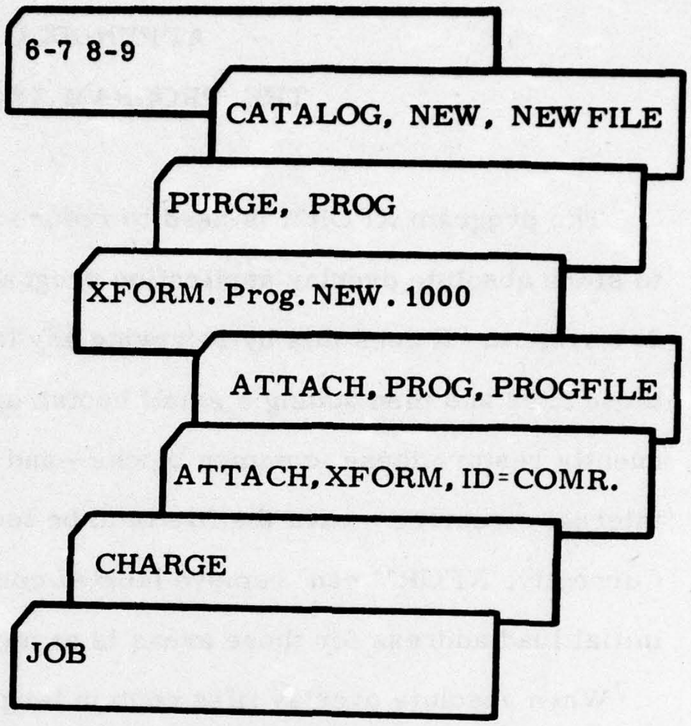

Figure 9 - Structure of the XFORM Job Deck

As implied in Figure 9 the format of the control card which executes the XFORM program is as follows:

XFORM. OVFILE/XFILE/NUM

where XFORM is the logical file name of the XFORM program

OVFILE	is the logical file name of the absolute overlay program file to be processed by XFORM
XFILE	is the logical file name of the file which has been transformed by XFORM; XFILE is automatically placed on a permanent file device and will contain the bootstrap program required for future execution.
NUM	is the octal number of words of labeled common to be removed from OVFILE during XFORM's transformation process. If no values have been preset in the labeled

common (i.e., no values set by BLOCK DATA subprograms), the entire labeled common may be removed; otherwise, only labeled common up to the first variable defined through a DATA statement can be removed without possible error conditions. The user can maximize the removal of labeled common by grouping all of the labeled common blocks not having preset values ahead of those blocks having preset values. If NUM is less than zero, all of labeled common is to be removed. If NUM is not specified, a value of 20000B is assumed. If neither negative nor blank, the minimum NUM value specified must be at least 60 octal words to allow space for the bootstrap loader program. The special characters

, . ( )

may be used instead of the slash.

XFORM, which can accommodate any size and number of overlays, removes only the first NUM words of labeled common from the main overlay.

When XFORM has completed its processing, the XFILE may be cataloged for future reference. Whenever XFILE is executed, the SCOPE system loader will load and execute the bootstrap for XFILE, which in turn will load the main overlay into memory. The bootstrap displaces the main overlay by NUM words in order to reestablish the correct program structure before giving control to the main overlay and dropping out of memory.

REFERENCES

1. Rhodes, T., "The Computer-Aided Design Environment (COMRADE) Project," paper presented at the 1973 National Computer Conference, New York, American Federation of Information Processing Societies (Jun 1973).

2. Brainin, J., "Use of COMRADE in Engineering Design," paper presented at the 1973 National Computer Conference, New York, American Federation of Information Processing Societies (Jun 1973).

3. Tinker, R. and I. Avrunin, "The COMRADE Executive System," paper presented at the 1973 National Computer Conference, New York, American Federation of Information Processing Societies (Jun 1973).

4. Chernick, C., "The COMRADE Design Administration System," paper presented at the 1973 National Computer Conference, New York, American Federation of Information Processing Societies (Jun 1973).

5. Willner, S., et al, "COMRADE Data Management System," paper presented at the 1973 National Computer Conference, New York, American Federation of Information Processing Societies (Jun 1973).

6. Wallace, M. and A. Bandurski, "COMRADE Data Mangement System Storage and Retrieval Techniques," paper presented at the 1973 National Computer Conference, New York, American Federation of Information Processing Societies (Jun 1973).

7. Control Data Corp., "INTERCOM Reference Manual," Publication No. 60307100.

8. Control Data Corp., "COMPASS Reference Manual," Publication No. 60360900.

9. Control Data Corp., "Loader Reference Manual," Publication No. 60344200.

10. Control Data Corp., "FORTRAN Extended Version 2 Reference Manual," Publication No. 60305600.

11. Control Data Corp., "SCOPE Reference Manual," Publication No. 6030//200.

12. Control Data Corp., "System Programmers Reference Manual," Publication No. 60306500.

13. Martin, R., "Partitioned Data Set Utility Routines for the Control Data CDC-6700 Computer," Naval Ship Research and Development Center Report 4354 (Apr 1974).

INITIAL DISTRIBUTION

Copies	
1	U.S. Army Electronics Command/AMSEL-GG-CG Robert Dunn
1	CHONR 437/Denicoff
1	NAVSEA 6E24/Chaiken
25	NAVSEC:
	1 SEC 6105B/Dietrich
	20 SEC 6105B/Berry
	1 SEC 6114B2/Fuller
	1 SEC 6133E/Straubinger
	1 SEC 6152D/Lee
	1 SEC 6179A20/Singer
12	DDC
1	NASA-Langley/R. Fulton
1	Grumman Aerospace Corp./William H. Mueller

CENTER DISTRIBUTION

Copies		Copies	
1	18/1808 Gleissner	1	189/Gray
1	1802.2/Frenkiel	1	1882.1/Strickland
1	1802.4/Theilheimer	30	5214.1 Reports Distribution
1	1805/Cuthill	1	5221 Library--Carderock
2	1809/Harris	1	5222 Library--Annapolis
20	1832/Gorham	1	712.9/Pierce
1	1834/Zaritsky		
1	1835/Culpepper		
1	184/Lugt		
1	185/Corin		
1	1851/Brainin		
1	1853/Thomson		
1	1854/Lynch		
1	1855/Brengs		
1	1856/Skall		
1	186/Sulit		
1	188/Camara		
1	1882/Wallis		